بسم الله الرحمن الرحيم

12/02/65

นิพจน์

Expressions

นิพจน์ (Expressions)

นิพจน์ (Expression) คือ การดำเนินการที่ทำให้ได้ผลลัพธ์เป็นหนึ่งค่า การใช้นิพจน์มีองค์ประกอบ 2 ส่วน คือ

1. ตัวดำเนินการ (operator)
2. ตัวถูกดำเนินการ (operand)

นิพจน์เลขคณิต (Arithmetic Expression)

Arithmetic Expression	ความหมาย	ตัวอย่าง	ผลลัพธ์(สมมุติให้ a=8 และ b=4)
+	การบวก	6 + 2 a + b	8 12
-	การลบ	6 - 2 a - b	4 4
*	การคูณ	6 * 2 a * b	12 32
/	การหาร	6 / 2 a / b	3 2
%	การหาเศษจากการหาร	6 % 2 a % b	0 0

นิพจน์ทางตรรกะ (Boolean Expression)

นิพจน์ทางตรรกะเป็นนิพจน์ที่มีค่า 2 ค่า คือ จริงหรือเท็จ กำหนดได้จากค่าจำนวนเต็ม 1 แทน จริง และค่า 0 แทน เท็จ นิพจน์ทางตรรกะได้มาจากการใช้ตัวดำเนินการ 2 แบบ คือ

1. ตัวดำเนินการสัมพันธ์ (relational operator)
2. ตัวดำเนินการตรรกะ (logical operator)

ตัวดำเนินการสัมพันธ์ (Relational Operator)

คือ การเปรียบเทียบระหว่างข้อมูล 2 ตัว โดยผลลัพธ์จากนิพจน์ที่ใช้ตัวดำเนินการสัมพันธ์จะได้ค่าเป็นจริง (1) หรือเท็จ (0)

Relational Operator	ความหมาย	ตัวอย่าง	ค่าที่ได้เมื่อกำหนดให้ x=12 และ y=-5
>	มากกว่า	x > y	1
>=	มากกว่าเท่ากับ	x >= y	1
<	น้อยกว่า	x < y	0
<=	น้อยกว่าเท่ากับ	x <= y	0
==	เท่ากับ	x == y	0
!=	ไม่เท่ากับ	x != y	1

ตัวดำเนินการทางตรรกะ (Logical Operator)

นิพจน์ทางตรรกะหลายประโยค สามารถนำมาใช้ร่วมกันได้โดยการเชื่อมกันด้วยตัวดำเนินการทางตรรกะ (Logical Operator) ที่จะให้ได้ผลลัพธ์ออกมาเป็นค่าทางตรรกะ จริงหรือเท็จ

Logical Operator	ความหมาย
&&	และ (AND)
||	หรือ (OR)
!	นิเสธ (NOT)

ตารางสรุป และ หรือ นิเสธ

A	B	A && B	A || B	!A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

ความหมายของตัวดำเนินการสัมพันธ์และตัวอย่างการใช้

คำสั่งกำหนดค่า (Assignment Statement)

การใช้คำสั่งกำหนดค่าให้กับตัวแปร สามารถใช้ตัวดำเนินการได้ 2 รูปแบบ

1. ตัวดำเนินการกำหนดค่า (Assignment Operator)
2. ตัวดำเนินการประกอบ (Compound Operator)

1. ตัวดำเนินการกำหนดค่า (Assignment Operator)

การกำหนดค่าให้กับตัวแปรใช้เครื่องหมาย = เป็นตัวดำเนินการกำหนดค่าในรูปแบบ

```
var_name = constant ;     หรือ     var_name = expression ;
```

var_name : ชื่อตัวแปรที่ถูกกำหนดค่าและต้องอยู่หน้าเครื่องหมาย = เสมอ

constant : ค่าคงที่

expression : นิพจน์

ตัวอย่าง

```
x = 1 ;
```

หรือ

```
x = y + 1 ;
```

ตัวอย่างคำสั่งกำหนดค่า

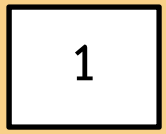
```
1.   number  =  1 ;
2.   number  =  number  +  5 ;
```

การทำงานของโปรแกรม

number = 1

number = number + 5

ค่าของตัวแปรในหน่วยความจำ



number

number

โปรแกรมที่ 1 การเพิ่มค่าจำนวนเต็ม

```
1.  #include<stdio.h>
2.  int  main()
3.  {
4.          int x = 5 ;
5.          x = x + 10 ;
6.          printf("x = %d\n",x);
7.          return 0;
8.   }
```

15

cout : x = 15

| เว้นวรรค

โปรแกรมที่ 2 การหาพื้นที่วงกลม โดยรับค่ารัศมีจากผู้ใช้

```
1.   #include<stdio.h>
2.   #define PI 3.14159
3.   int  main()
4.   {
5.         float radius ;
6.         float area ;
7.         printf(“Please enter radius : ”);
8.         scanf(“%f”,&radius);
9.         area = PI * radius * radius ;
10.        printf(“Area of the circle = %.2f ”,area);
11.        return 0;
12.   }
```

2. ตัวดำเนินการประกอบ (Compound Operator)

การกำหนดค่าให้กับตัวแปร

Compound Operator	ตัวอย่างการใช้งาน	ความหมาย	ค่าของ x
+=	x += y	x = x + y	7
-=	x -= y	x = x - y	3
*=	x *= y	x = x * y	10
/=	x /= y	x = x / y	2.5
%=	x %= y	x = x % y	1

สมมติกำหนดให้ x = 5 และ y = 2

ตัวดำเนินการแบบยูนารี (Unary Operator)

ตัวดำเนินการยูนารี เป็นตัวดำเนินการที่มีจำนวนของตัวถูกดำเนินการ 1 ตัว

- **ตัวดำเนินการเพิ่มค่าและลดค่า (Increment and Decrement Operator)**

ใช้เครื่องหมาย ++ สำหรับการเพิ่มค่า และ -- สำหรับการลดค่า

การใช้งานมี 2 รูปแบบ

1. **prefix mode** ตัวดำเนินการไว้ข้างหน้า
2. **postfix mode** ตัวดำเนินการไว้ข้างหลัง

การคำนวณ	ตัวอย่าง	ความหมาย	ค่าของ a
เพิ่มค่าของตัวถูกดำเนินการอีก 1 (prefix mode)	++a	a = a + 1	3
เพิ่มค่าของตัวถูกดำเนินการอีก 1 (postfix mode)	a++	a = a + 1	3
ลดค่าของตัวถูกดำเนินการออก 1 (prefix mode)	--a	a = a - 1	1
ลดค่าของตัวถูกดำเนินการออก 1 (postfix mode)	a--	a = a – 1	1

สมมติกำหนดค่าเริ่มต้น a = 2

Precedence และ Associativity

- Precedence ลำดับการทำงานก่อนหลังของตัวดำเนินการเมื่อใช้ตัวดำเนินการมากกว่าหนึ่งตัว
- Associativity ลำดับการทำงานจากซ้ายไปขวาเมื่อตัวดำเนินการมี Precedence ในลำดับเดียวกัน

Precedence	กลุ่มของตัวดำเนินการ	ตัวดำเนินการ operator	Associativity
1	เครื่องหมายวงเล็บ	( ) [ ]	ซ้ายไปขวา
1	เครื่องหมายเรียกใช้ชนิดข้อมูลแบบโครงสร้าง (structure)	. ->	ซ้ายไปขวา
2	ตัวดำเนินการ unary	- ++ - ! * & (type) sizeof (type)	ขวาไปซ้าย
3	คูณ หาร และหาเศษส่วนจากการหาร	* / %	ซ้ายไปขวา
4	บวกและลบ	+ -	ซ้ายไปขวา
5	Bitwise Shift Right , Left	>> <<	ซ้ายไปขวา
6	ตัวดำเนินการสัมพันธ์	< <= > >=	ซ้ายไปขวา

precedence และ associativity ของ operator

Precedence	กลุ่มของตัวดำเนินการ	ตัวดำเนินการ operator	Associativity
7	ตัวดำเนินการสัมพันธ์	== !=	ซ้ายไปขวา
8	Bitwise	&	ซ้ายไปขวา
9	Bitwise Exclusive OR	^	ซ้ายไปขวา
10	Bitwise OR	|	ซ้ายไปขวา
11	ตัวดำเนินการทางตรรกะ AND	&&	ซ้ายไปขวา
12	ตัวดำเนินการทางตรรกะ OR	||	ซ้ายไปขวา
13	ตัวดำเนินการเงื่อนไข	?:	ขวาไปซ้าย
14	ตัวดำเนินการกำหนดค่า	= += -= *= /= %=	ขวาไปซ้าย
15	Comma	,	ขวาไปซ้าย

ตัวอย่าง

ลำดับการทำงานแบบ Precedence

k = 10 + 2 * 4

ในกรณีนี้ Precedence ของ * เหนือกว่า + เพราะฉะนั้นค่าของ k ได้มาจาก 2 * 4 เป็น 8 แล้วนำไปบวกกับ 10 จึงเท่ากับ 18

ลำดับการทำงานแบบ Associativity

10 / 2 * 4 % 3 *4

ตัวดำเนินที่ใช้ คือ / * % ซึ่งมี Precedence ในลำดับเดียวกัน ทำให้ได้ Associativity ที่ทำงานจากซ้ายไปขวา จึงเท่ากับ

((((10 / 2) * 4) % 3) * 4)

ตัวอย่างการทำงานจากขวาไปซ้าย

a = b = c = d = 0

การแปลงชนิดข้อมูล (Type Conversion)

ในภาษา C มีการแปลงชนิดข้อมูล 2 รูปแบบคือ

1. implicit type conversion
2. explicit type conversion

1. Implicit Type Conversion

นิพจน์ที่ประกอบด้วยตัวถูกดำเนินการตั้งแต่สองตัวขึ้นไปที่มีชนิดข้อมูลแตกต่างกัน ค่าของนิพจน์ยึดเกณฑ์การแปลงข้อมูลไปเป็นชนิดที่ขอบเขตใหญ่กว่า เรียกการแปลงชนิดข้อมูลแบบนี้ว่า implicit type conversion

โปรแกรมที่ 4.3 การแปลงชนิดข้อมูลแบบ implicit type conversion

```
1.  #include<stdio.h>
2.  int main()
3.  {
4.      float floatNum = 25.21;
5.      int intNum = 300;
6.      short shortNum = 10;
7.
8.      printf("int * short is int : %d\n", intNum * shortNum);
9.      printf("float + int is float : %.2f\n", floatNum +intNum);
10.     return 0;
11. }
```

โปรแกรมที่ 4.4 การแปลงชนิดข้อมูลแบบ implicit type conversion

```
1.  #include<stdio.h>
2.  int  main()
3.  {
4.      int  iValue   =  10;
5.      float  fValue  =  15.5;
6.      float  fResult;
7.
8.      fResult  = iValue  +  fValue;
9.      printf("%d  +  %.2f  =  %.2f",  iValue,  fValue,  fResult);
10.     return 0;
11. }
```

Explicit Type Conversion

กรณีต้องการแปลงชนิดข้อมูลของผลการคำนวณของนิพจน์หรือค่าของตัวแปรให้เป็นชนิดข้อมูลที่ต้องการเรียกการแปลงชนิดข้อมูลแบบนี้ว่า explicit type conversion ซึ่งมีรูปแบบดังนี้

(type) variable; หรือ **(type)** expression;

เช่น
```
int iNum = 5;
float fNum;
fNum = (float) iNum;
```

ตะแบบ๑๐

โปรแกรมที่ 4.5 การแปลงชนิดข้อมูลแบบ explicit type conversion

```
1.    #include<stdio.h>
2.    int  main()
3.    {
4.          int  i1  =  10;
5.          int  i2  =   3;
6.          float  f;
7.
8.          f  =  i1  /  i2 ;
9.          printf("No casting  i1 / i2  ->  f  =  %.2f\n",  f );
10.         f   =  (float)  (i1 /  i2);
11.         printf("casting  by  (float)  (i1  /   i2) ->  f  =  %.2f\n",  f);
12.         f  =  (float)   i1 /i2;
13.         printf("casting  by  (float)   i1  /  i2 ->  f  =   %.2f\n",   f);
14.         return 0;
15.   }
```